Universitat de Girona
Escola Politècnica Superior

Pràctiques ESII

Curs 2023/24

GEINF- GDDV

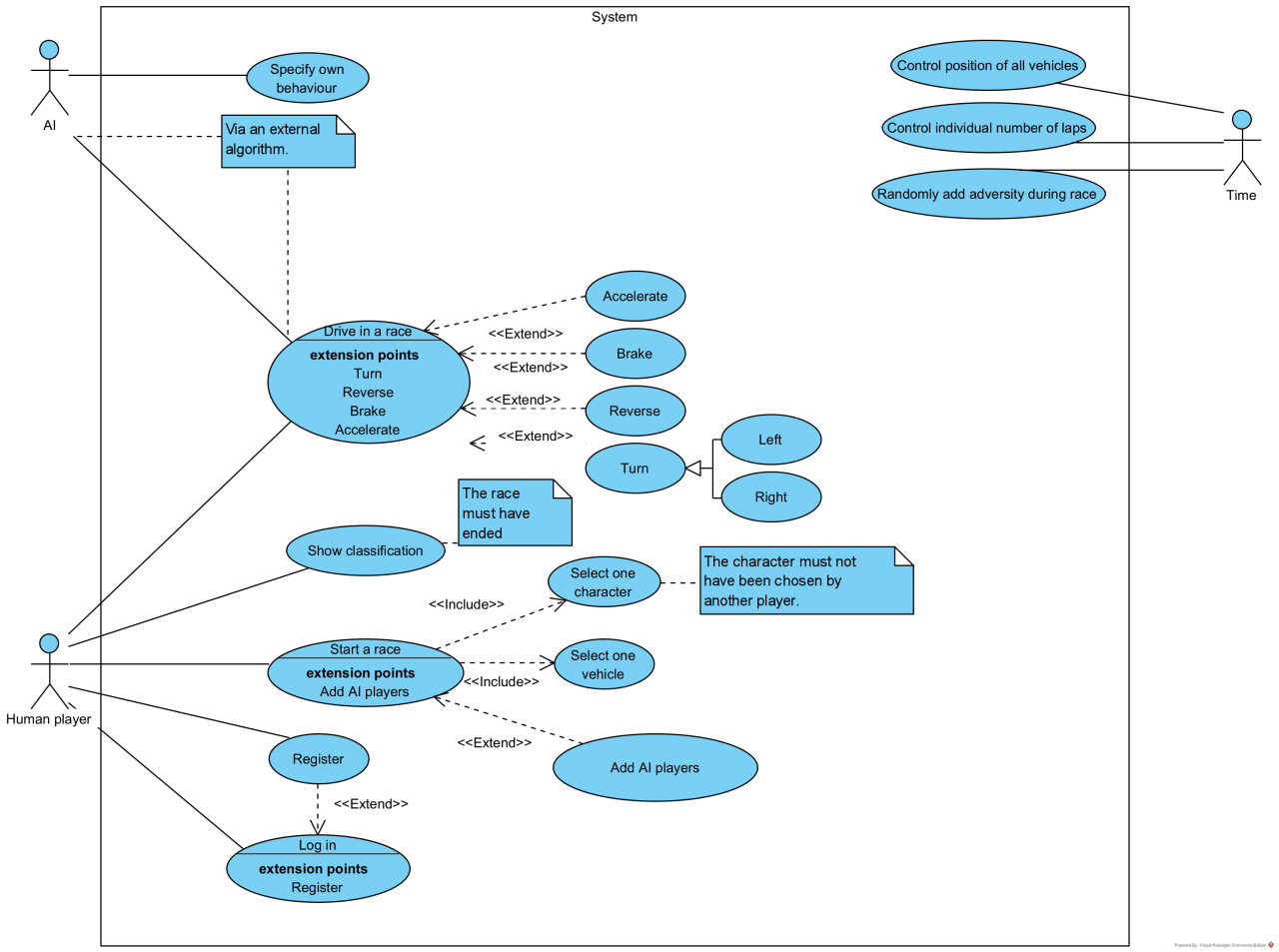
Pràctica 1

Grup T
Jordi Badia, Aniol Juanola i Guillem Vidal

(October 22, 2023)

Contents

1 Use case diagram



1.1 Use case sheet “Drive in a race”

Drive in a race

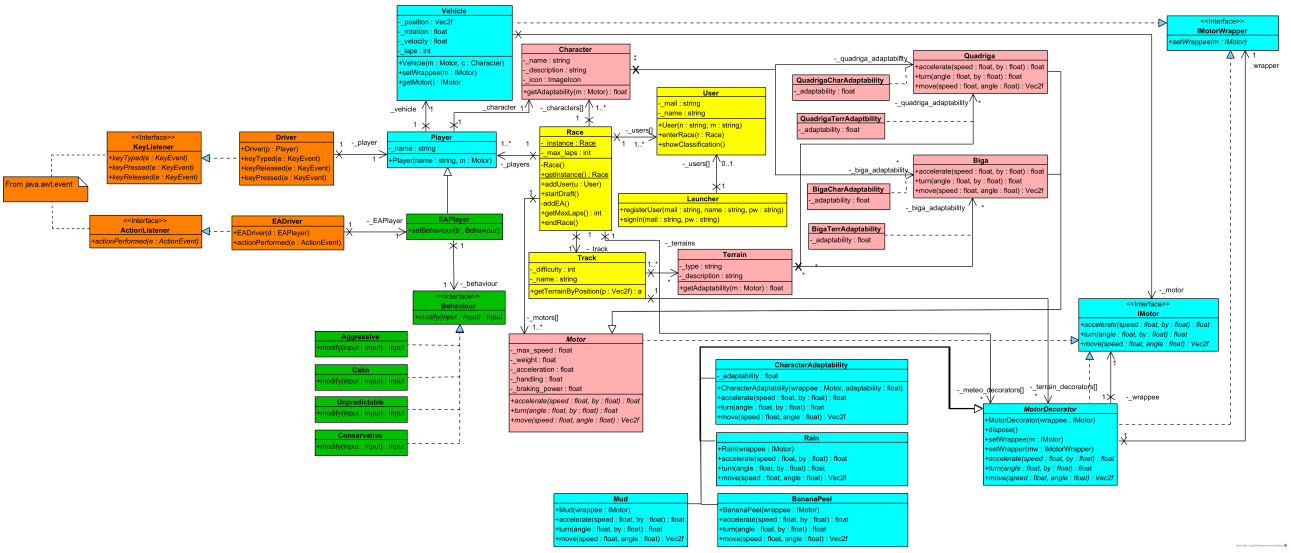
Field	User vision
Description	Use case in how the human player is expected to move the vehicle during a race.
Main actor	Human player, AI
Precondition	• The actor has selected a character and vehicle. • The race has started and has not ended. • The input method has been chosen.
Main flow	1. Check whether the action is valid (accelerate, brake, turn left, turn right). 2. The player presses the accelerate action key/button. 3. The vehicle accelerates while the key remains pressed. 4. Once the key is released, the vehicle stops accelerating.
Alternative flows	1a. The action is not valid or not described. 2. The vehicle brakes while the key remains pressed. 2a. The player presses the brake action key/button. 3. The vehicle brakes while the key remains pressed. 4. Once the key is released, the vehicle stops braking. 2b. The player presses the turn left action key/button. 3. The vehicle fully turns left while the key remains pressed. 4. Once the key is released, the vehicle stops turning. 2c. The player presses the turn right action key/button. 3. The vehicle fully turns right while the key remains pressed. 4. Once the key is released, the vehicle stops turning.
Postcondition	The input has been read and the player’s vehicle has moved accordingly.
Non-functional requirements	The input delay is low enough to make the game “playable” The connection is stable in order to compete online with other players and synchronize their positions. The hardware the game is running on is equal or better than the lowest requisites.

1.2 Use case sheet “Show classification”

Show classification

Field	User vision
Description	Use case in how the system will output the classification of the race once a player asks for it.
Main actor	Human player
Precondition	• The actor has participated in the race. • The race has concluded. • The actor has pressed the button to show the classification.
Main flow	1. The system fetches all the players names and race times. 2. The system sorts the players names by remaining distance to the finish line in ascending order. 3. The system fetches each player’s character and vehicle. 4. A table is built with the player’s name, character, vehicle and overall distance and lap count. 5. The table is shown on the screen.
Alternative flows	1a. A player has abandoned or disconnected during the race. 2. Their time will be set to Infinite and it will be shown as “Disqualified”.
Postcondition	The scoreboard is shown on the actor’s screen.

2 Class diagram



Green strategy pattern to determine EA's behaviour. The function `Modify(Input)` alters the EA's intention and `setBehaviour(Behaviour)` changes the behaviour itself.

Orange interfaces and their implementations regarding player input.

Yellow miscellaneous classes.

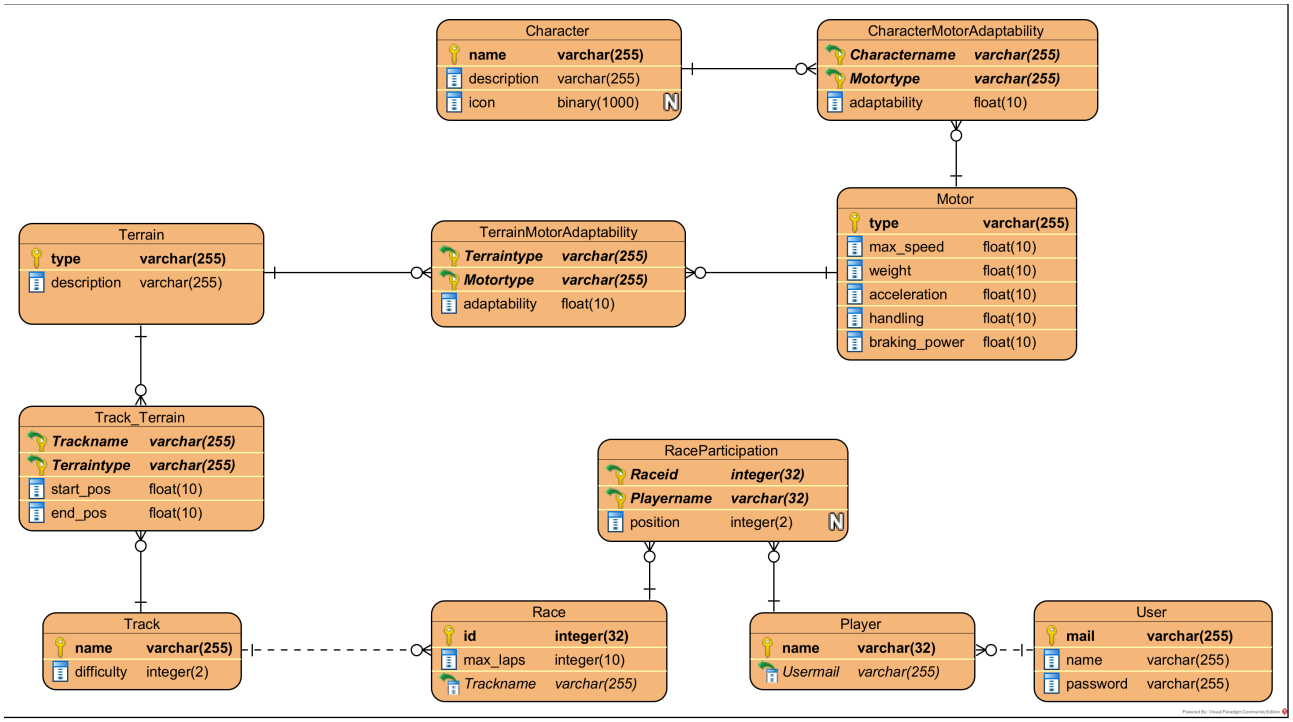
- Race can be understood as the lobby of the execution. One can add users to the race, and the remaining space will be filled by EAs. Then, all players must choose both the character and the vehicle they are going to play with. And with that the race can begin. Race manages the instances of motor and character, which are unchanging and shared between players. When a vehicle has completed a `_max_laps` number of laps, the race ends.
- User contains the information of the client (name, email...).
- Launcher is the client: log ins and sign ups go through this class. The password is not stored in the User class, instead Launcher uses it to identify the user in the database.

Pink constant classes of characters, motors, terrains and adaptabilities.[1]

Blue all the classes that store the current state of the race in relation to each player (position, rotation, velocity). Effects are applied on vehicles via decorators, either character or terrain adaptabilities, atmospheric conditions or in-game items (such as banana peels or mushrooms). In order to erase an effect, decorators, being linked lists, must be doubly linked. The `IMotorWrapper` interface allows decorators to be treated as nodes in a list, and vehicle, being an end point of the list, implements it as well.

[1] Adaptability classes will be implemented as maps.

3 EER diagram



This diagram represents what one would store in a real database, i.e. there's no information about the current state of the game but its persistent data.